HOMEWORK #5:

No Parenthesis Necessary

Due Thursday, October the 31th, 11:59pm

For this assignment, you will submit a single C++ file called '`robotcalc.cpp`'.
Remember to put your name and section at the top of your program file.
Your program should expect all input to come from '`cin`', and all your output should be to '`cout`'.

Problem

We all know that robots are superior to humans. One way in which robots are better than humans is in the way they evaluate arithmetic expressions. Instead of using *in-fix* notation like humans do, robots use *post-fix* notation, in which the operator follows the operands. For example, while a human would write:

```
5 + 4
```

A robot would write:

```
5 4 +
```

This notation allows robots to avoid wasting precious memory with silly parentheses and baroque operator precedence. While humans needs parenthesis to indicate the order of evaluation like in:

```
7 * (5 - 3)
```

Robots simple follow the order of operations from left ro right, as in :

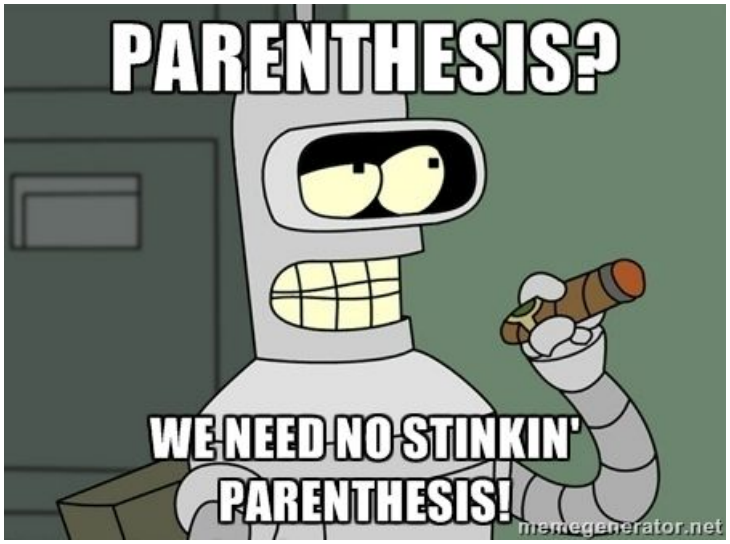
```
5 3 - 7 *
```

or alternatively:

```
7 5 3 - *
```

This super-human ability is possible because robots organize their short term memory as a **stack**. When robots read an *operand* they push it into their short-term memory stack. When they read an *operator*, they pop the necessary operands and push the intermediate result into the **stack**.

Your job is to write a program that simulates the way robots process integer expressions and create a "Robot Calculator". Your program will use a stack to simulate a robot's short term memory.



(Typical Robot)

Input

The input will consist of a series of integer expressions in *post-fix* notation. Elements of the expression are separated by spaces.
You will implement the following robot integer expression operators:

- Binary operators '`+`', '`-`', '`*`', '`/`', '`%`' with their usual meanings.
- Unary operator '`!`' negation... (Example... `5 !` produces `-5`)
- Aggregate operator '`sum`' : the sum all the elements in the stack.
- Aggregate operator '`prod`' : the product of all the elements in the stack.
- Stack operator '#' : prints the content of the stack.
- Stack operator '$' : clears the stack.

Character '@' will denote the end of the input.

Output

Simulate a robots expression evaluation and output the contents of the stack (formated as in the sample) whenever the '#' command is found.
REMEMBER: When your program reads an operator, it should pop the corresponding number of operands off the stack, apply the operator, and push the result back into the stack.

Implementation Requirements / Details.

- Use a **stack** Data Structure to simulate a robot's computational processes.
- Your stack implementation should be a subclass of the provided "`AbstractStack`" class.
- Your '`main()`' function should be inside a file called '`robotcalc.cpp`'
- All operands are integers.

- An expression may take more than one line.
- All expression elements are separated by at least a single space.
- The stack is printed only when the '#' command is encountered.

.

Sample

Input	Output
1 2 3 # $ 4 3 * ! # 20 3 / # $ # 62 # $ 2 3 8 * + # 4 48 4 2 + / # sum # $ 1 2 3 4 5 # prod # @	[3, 2, 1] [-12] [6, -12] [ ] [62] [26] [8, 4, 26] [38] [5, 4, 3, 2, 1] [120]

Reference:

http://en.wikipedia.org/wiki/Reverse_Polish_notation

Hints:

Study the following code sample:

```
#include <string>
#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
  string s1, s2;
  int x, y;

  cin >> s1 >> s2;
  x = atoi( s1.c_str() );
  y = atoi( s2.c_str() );
  cout << s1 << " * " << s2 << " = " << x*y << endl;

  return 0;
```

```
}
```